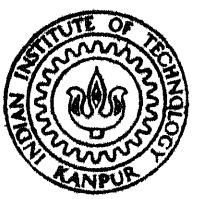
TABLE INTERFACE AND EDITOR FOR RELATIONAL DATABASE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

BY
JUGAL KISHORE MARWAHA

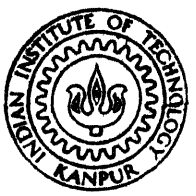


TO THE
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
FEBRUARY, 1985

TABLE INTERFACE AND EDITOR FOR RELATIONAL DATABASE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

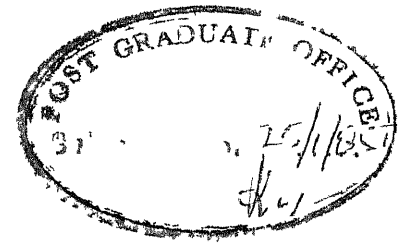
BY
JUGAL KISHORE MARWAHA



TO THE

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
FEBRUARY, 1985

EE-1985-M-MAR-TAB.

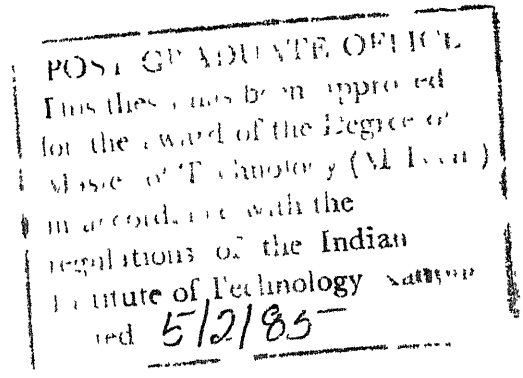


CERTIFICATE

This is to certify that the thesis entitled 'Table Interface and Editor for Relational Database' is a report of work carried out under **our supervision** by Jugal Kishore Marwaha and that it has not been submitted elsewhere for a degree.

(R. Raghuram)
Assistant Professor
Department of Elect.Engineering
Indian Institute of Technology
Kanpur.

(Rajeev Sangal)
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur.



ACKNOWLEDGEMENTS

I am extremely happy to record my indebtedness to Dr. R. Sangal and Dr. R. Raghuram for their inspiring guidance and suggestions throughout the course of this work.

I would also like to acknowledge my gratitude to V.M. Shrivastava who with me made the implementation of the system a success.

Thanks to all those friends who made my stay at I.I.T. Kanpur enjoyable and memorable. For the excellent job done in typing this manuscript, I thank Mr. C.M. Abraham.

JUGAL KISHORE MARWAHA

ABSTRACT

One of the widespread uses of computers is in storing and retrieving of large amounts of data. An intermediary (i.e., a programmer or a system specialist) is necessary before a user can use the computer. This increases response time, cost and inconvenience to the user. To address nonprogrammer community, we have implemented Table Interface and Editor system for relational database. The system is screen-oriented and interactive. It allows a user to define a table corresponding to a relation in a relational data base. The table is displayed on an ordinary terminal screen; graphic terminal is not required. The user can enter, view and modify data in a relation through the corresponding table.

The system allows one to define the tables corresponding to four relations at a time.Moreover, one can define four views (splittings) for a relation. The user is allowed to view more than one relation at a time on the screen. Horizontal and vertical scrolling helps in viewing the hidden data. We have implemented paths, a facility by which the Return key steps us through the cells in a predefined path. The facility for printing, saving into a file and loading from a file is also available.

my beloved

parents

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

One of the widespread uses of computers is in storing and retrieving of large amounts of data. As the computers provide fast access and easy manipulation of data, even the nonprogrammers have started using the computer for storing data.

A database is a collection of data stored in a computer. A Database Management System (DBMS) is the software which provides efficient storage and access to data, on one hand and an abstract view of the data, on the other hand. It allows one or more users to modify or retrieve the data using operators in the abstract view.

If a nonprogrammer user wishes to store and manipulate data, he normally seeks the help of a programmer or a systems specialist. This introduces an intermediary between the user and the computer. It results in greater

(i) response time in answering the queries of a user as the time will be consumed in developing and debugging the required programs first.

(ii) cost.

(iii) inconvenience caused due to unavailability of the specialist.

A better solution is to have a programmerless machine i.e., an easy to use interface. As it needs no programming so no intermediary will be there.

The solution is to display data through an interface and store, retrieve, update the data by an editor corresponding to the interface.

1.2 OBJECTIVES

The following are the objectives of our system.

1. To design an interface which helps in handling large amounts of data.
2. To design an Editor (corresponding to the interface) by which one can enter, modify or view the desired data.
3. As far as possible, the functions of various keys present on the keyboard must remain same as normal.
4. The interface should be easy to learn.

1.3 APPROACH AND SOLUTION

The data can be displayed and viewed through relational, network or hierarchical data models. The relational data model is simple, and there is a strong theoretical support

for it. The mathematical concept behind relational data model is the set theoretic relation. A relation consists of a set of tuples with each member having the same set of attributes. A relation with simple domain can be represented by the tabular structure as the attributes and tuples of a relation can be represented by columns and rows (of a table) respectively.

Achievements

Table interface and editor system based on the relational data model has been implemented. Here, the user is allowed to enter, display and edit data through a table. The table is displayed on the ordinary terminal screen i.e. graphic terminal is not required. Data is entered by first positioning the cursor in the desired cell of the table and then typing the characters. Thus, the system is screen-oriented. The system allows a user.

(i) To define a table in order to enter various tuples of a relation.

(ii) To move the cursor

(a) within a cell

(b) to any cell of the table. This kind of movement is done by arrow keys and return key. We have implemented paths, a facility by which the Return key steps us through the cells in the predefined path.

(iii) to display and modify a desired tuple. The tuple is displayed roughly in the middle of the table along with the adjacent data.

(iv) to browse through a relation by scrolling the table. Both vertical and horizontal scrolling have been implemented. If the size of table is small then it may display only a part of tuple. The right hidden or left hidden portions of a tuple can be displayed by typing moveright or moveleft arrow keys, respectively. (From the cell present at the corresponding edge of the table.)

(v) to define upto four tables representing the same relation in order to view desired attributes of the selected tuples present in a relation. These kinds of tables are termed as views (or splittings) of the relation.

(vi) to define upto four tables representing four different relations. At a time, we can define 16 tables in the present system (4 relations each having 4 views).

(vii) to display the desired number of tables on the screen as 16 tables may not be accommodated on the screen. The system manages the size and no. of tables, if the user unknowingly specifies the no. of tables which requires more space than the available space on the screen.

(viii) To alter the set of key attributes and modify their sequence of preference.

Brief summary lines are displayed on the screen. These display the given command and the corresponding dialogue (including the error messages in case of an error). Thus the system is interactive. It provides help at all times. The facilities for printing, saving into a file, loading from a file and redisplaying the screen are also incorporated in the system.

1.4 OUTLINE OF THE THESIS

The following is the organization of the thesis.

The first chapter gives an introduction to our thesis work.

Background for the work is discussed in the second chapter. In the first part, user interfaces and editors and then the file interface are discussed.

The third chapter deals with design considerations. The first part discusses the objectives in detail.

Secondly, the design choices are discussed. Lastly, the various commands and their corresponding messages are given in user manual.

The fourth chapter deals with the implementation. Data structures are given in the first part and program structures thereafter. Program structures explain the flow of controls and data in our system.

The fifth chapter concludes our thesis work. In the first part summary of the work is given. Then the limitations and fugure works are discussed.

CHAPTER 2

BACKGROUND

2.1 USER INTERFACES AND EDITORS

An Editor uses power of computer for creation, addition, deletion and modification of text material such as program statements, manuscript text and numeric data. The editors allows text to be modified and corrected many orders of magnitude faster and more easily than would manual correction. No longer are editors thought of as tools only for programmers, it is now increasingly realized that the editor should be considered as the primary interface to the computer for all type of 'knowledge-workers', as they compose, organize, study and manipulate computer based information. An interactive editor is a computer program that allows a user to create and revise a target dudocuments. The term document includes the targets such as computer programs,text, equations, tables,diagrams, line art and half tone or colour photographs anything that one might find on a printed page. The document editing process is an interactive user-computer dialogue. Following are some of the functions that are available on an editor :

(i) To select what part of the target is to be viewed and manipulated.

(ii) To determine how to format this view on-line and then to display it.

(iii) To specify and execute operations that modify the target document.

(iv) To update the view appropriately.

The user interface for an editor implies the collection of tools and techniques with which the user communicates with the editor. It contains the input devices, the output devices and the interaction language of the system.

(1) INPUT DEVICES - These are used for three main purposes :

(i) To enter elements

(ii) To enter commands

(iii) To designate editable elements.

These devices as used with editors can be divided into various categories as :

(a) TEXT DEVICES - These are typically typewriter like keyboard on which a user presses and releases a key to send to the CPU or to an I/O controller a unique code for each key.

(b) BUTTON OR CHOICE DEVICES - They generate an interrupt or set a system flag, usually causing invocation of an associated application program action. Alternatively, buttons are often simulated in software by having the user choose text strings or symbols displayed on the screen.

(c) LOCATOR DEVICES - These are the x-y Analog-to-Digital transducers that position a cursor symbol on the screen by continually sampling the analog values produced by the user's movement of the device.

(d) VOICE INPUT DEVICES - These are still in research stage. It translates spoken words - both, literal text and commands to their textual equivalents.

(2) OUTPUT DEVICES - These serves to let the user view the elements being edited and the results of the editing operations. CRT terminals and hard-copy printers are the main output devices.

(3) INTERACTION LANGUAGE - It mainly has three components.

(a) LEXICAL COMPONENT - It specifies the way in which lexemes, information from the input devices or for the output devices, are combined to form the tokens used by the syntactic component.

(b) SYNTATIC COMPONENT - It specifies the input and output rules by which tokens are put together to form sentences in the grammer of the language. In terms of editors, the set of tokens might include character strings, commands and screen positions. In terms of output, the set of tokens might include character strings, lines and formatted paragraphs.

(c) SEMANTIC COMPONENT - These specify functionality. What operations are valid for each element, what information is needed for manipulation of each element. What the results of operation are and what errors may occure.

In an editor system, the command language processor accepts input from the user's input devices lexically analyzes the input stream of characters and generates a stream of tokens, syntatically analyzes the accumulated stream of tokens and upon finding a legal composition of tokens, invokes the appropriate semantic routines. Finally, the semantic routines invoke any of the user operation.

The user operations fall into several subcategories. Traveling operations allow the user to browse through document. Viewing operations allow the user to control what subset of target data is presented to the user and how it is formatted. Edit allow him to manipulate target elements.

(1) Traveling can be a simple movement i.e. position dependent traveling or it can be by pattern searching i.e. content dependent specification of location. Some editors provide the facility to travel between 2 or more files in on-line authoring i.e. interfile motion.

(2) The viewing component creates an up-to-date view on the display. Viewing operations are of 3 sorts -

(i) filtering operations, in which appropriate portions of the raw data structure of the file are selected for the viewing buffer.

(ii) operations that format these filtered data to produce an ideal view.

(iii) operations that map this ideal view to a window or page on a physical output device.

(3) The editing operation mainly involves :

(i) Creating - It involves the creation of a document. Display editors generally offer one of two kinds of input styles; type over mode or insert mode. In typeover mode, each typed character replaces the character at which the cursor is pointing.

In insert mode, each time a user types a character, the character at the cursor position and all those to its right are shifted right by one character and the typed character is inserted at the cursor position. This insertion can always be done without any commands.

(ii) DELETING - The delete command allows the user to delete the character present at the current position of the cursor.

(iii) CHANGING - The simplest change is the replacement of one letter with another. In typeover mode of a display editor, the correction is made by simply typing the new character over the erroneous one. In insert mode of a display

editor, the correction is made by typing the new character and then using a delete-char command to delete the old character which has been pushed to the right by the insert of a new character.

2.2 VARIOUS EDITORS

Editors can be differentiated from the view point of target applications for which they were designed, the elements and their operations, the nature of the interface and the system configuration. These categories don't form strictly independent axes. The choice of one frequently influences the choice of another.

A text editor is one of the basic components of a text processing system, which is concerned not only with creation and maintenance but also with formulating and interactive presentation of text. Program editors operate on programs, whether represented in textual form or in another canonical form such as a parse tree. Picture or graphics editor facilitates the creation and revision of computer based graphics.

A new development in the text processing field is the document preparation system, which integerates text editing; picture editing and formating. A voice editor is a specialized interactive editor in which the target is digitally encoded voice. Some more editors systems are as follows :

(1) Line Editors : In 1960s, interactive line editors were designed that allowed the user to create and modify disk files line by line. They have problem of inability to edit a string crossing line boundaries, and inability to search for a pattern crossing line boundaries.

The following are common examples of Line Editors.

(A) IBM's CMS Editor : It is a fixed-length line oriented editor with a textual interface, designed for a time-sharing system in which terminals lack cursor motion keys and function keys. It presents the user with a one line editing buffer (the amount of the document that can be edited at a given time), although this is extended for some operations. It, also, presents the corresponding one-line viewing buffer (the amount of the document that is used to construct the display). The display is the simple mapping of one-line viewing buffer to one-line window. The following are the salient features of this editor.

(i) It has Edit and Input modes. Edit mode gives the user access to all the functional capabilities of the editor, including the capability to switch to input mode. Input mode, however, only gives the user two options : typing in text, which is simply inserted into the file at the current line pointer or pressing dual carriage returns, which returns the user to edit mode.

(ii) It provides the ability to set-up logical tab stops - tabs implemented in the editor software rather than in terminal hardware - so that tabs may be specified by typing a user - chosen logical tab character in the input stream.

(B) SOS : The editing buffer defaults to one line, although for most SOS commands a user can specify a line number or range of line numbers to expand this editing buffer. The default viewing buffer is a line.

The salient features of SOS are given as follows :

(i) The major unit of manipulation is the line.

(ii) The commands are typed in prefix notation (Verb/noun) unlike the CMS editor.

(iii) It attaches fixed, visible line numbers to each line in a file being edited.

(iv) It allows the user to create logical pages within a file for selection and organization purposes.

(v) It has a regular-expression pattern-searching facility.

(vi) The following seven different modes of operation are possible in SOS Editor.

(a) Input mode, in which SOS accepts the text being typed and inserts it into the file,

(b) Read-only mode, in which a user can travel through a file but not modify it;

(c) Edit mode, in which a user performs editing, traveling and viewing operations;

(d) Copy-file mode, in which the user can copy part or all of a file into another one;

(e) Alter mode, in which a user can perform character-by-character intraline editing without pressing carriage return to execute the command;

(f) Alter/insert mode, in which the user can insert characters such as control characters that have special meaning to the editor;

(g) Decide mode, in which the user can make case-by-case decisions for substitute commands.

(C) UNIX ed : It is a variable-length line editor. It has a single-line viewing buffer but, like SOS, ed allows the user to expand the editing buffer for a command by specifying a range of line numbers in the form starting, ending as an optional prefix to each command. The following are the salient features of ed.

(i) Like the CMS Editor, ed has two main modes edit mode and append mode.

(ii) As in the CMS Editor, lines in ed have varying internal numbers. Thus traveling is done as in the CMS Editor, with both absolute and relative specifications and ith context pattern specification as well.

(iii) It provides a facility for user-specified regular expressions in patterns defining the scopes of operations (as opposed to other editors, which use regular expressions simply for search commands).

(iv) It has the ability to reference the scope of an operation indirectly in another operand of that operation.

(2) Stream Editors : They solved the problems of Line Editors by eliminating the boundaries altogether. They act upon a document as a single conteneous chain of characters as if the entire document were a single indefinitely long character string, rather than act upon fixed length or variable length lines. TECO, described below, is the popular editor of this category.

(A) TECO : TECO, the Text Editor and Corrector is an interpreter for a string processing language. The editing buffer is the amount of the file in memory. The viewing buffer on the document defaults to the null viewing buffer. The document is displayed only on the explicit command; the user can specify a viewing buffer of any size. The following are the salient features of TECO.

(i) Fundamental commands such as insert, delete and context search are available.

(ii) It supports commands for conditional execution to aid in creating more complex commands. Q-registers are available for holding any numeric or string value.

(iii) Although TECO is character oriented, special commands allow the user to edit a document in terms of a line model.

(3) Display or Cursor Editors : These allow a cursor to be moved anywhere on the text displayed on the multiline screen, i.e., a user is able to move the cursor to point to the text, he wishes to manipulate. Text is conceived of as a quarter - plane extending indefinitely in width and length, with the topmost, leftmost character the origin of the file. The user travels through this plane by using cursor keys and changes characters by overtyping. Text is input on the screen at the position of the cursor. The editing and viewing buffers can be moved left and right and multiple windows support easy interfile editing. These make use of pick and delete buffers.

The salient features of these editors are given below :

(i) It is modeless, since all typing on the screen is considered text, commands must be entered either through function keys, control characters and escape sequences, or by moving the cursor to and typing in a special command line at the bottom of the screen.

(ii) The command syntax is single operand postfix.

(iii) It has a marking facility which allows the user to select with the cursor two arbitrary points in the text to define a scope not easily specified with the element modifiers.

(4) Syntax Directed Editors : These editors have the knowledge of the syntax of the thing (e.g., a pascal program) being edited. These are structure editors that ensure that the structure always is constrained to preserve syntactical integrity.

We have implemented a table interface to the database. So, the kind of editor present in our system is the table editor. It is an interactive editor and allows the user to enter, display and modify the data.

The data is stored in the database by the 'ACCESS METHOD INTERFACE'. The interface provides the functions to access the data in the database efficiently and uniformly.

2.3 ACCESS METHOD INTERFACE

This interface is based on the Relational Data Model. AMI is the interface between the user and the file structures providing the structural transparence, i.e., it gives the user an independent view of the data, free from the underlying file structure. This is a set of standard functions provided to the user to gain access to the data stored in the database. Following are the various functions.

1. OPENR (Chno, Fname)

This function opens the file name specified in Fname and assigns a channel no. to it, which is used in other routines of AMI. For any legal operation on the database, the relation has to be opened. This sets up all the control structures for the corresponding file.

2. GETNEXTRECORD (Chno, Adrs)

This function retrieves the next record from the file with channel no. chno and put it at the location starting with Adrs.

3. PUTNEXTRECORD (Chno, Adrs)

This function writes at the next position in the file with channel No. Chno taking the record from location starting with Adrs.

4. GETRECORD (Chno, Adrs, Keyval)

This function retrieve the record with key-value keyval in the database and if found, puts it at the place pointed to by Adrs.

5. INSERTRECORD (Chno, Adrs)

This function inserts a record in a file with channel no. Chno taking the record stored starting from Adrs. It calculates the Keyvalue from the record itself and doesn't allow duplicate records.

6. DELETERECORD (Chno, keyval)

This function deletes a record with key-value keyval from a file with channel no. Chno.

7. UPDATERECORD (Chno, Adrs)

This function updates a record (only for non key fields) of a file with channel no. Chno byttaking the record from location starting with Adrs.

8. MODIFYRECORD (Chno, Adrs)

This function modifies a record (modifies key fields also) of a file with channel no. Chno by taking the record from location starting with Adrs.

9. CLOSER (Chno)

This function closes the file with channel no. Chno for any further operation on it.

10. CREATER (Fname, Flag, Atrcount, Adrs1, Adrs2)

This function creates a file name specified in Fname. The value of Flag 0 or 1 determines the HASHED or ISAM file structure respectively. Total no. of attributes are specified in Atrcount. Adrs1 and Adrs2 specify the no. of non key attributes and key attributes respectively.

As the various AMI functions are presently available for CP/M CROMEMCO system and expected to be available for DEC-10 system near MAY, 1985. So, in the present interface, we simulated the above functionality routines of our design. Here, some portion of main memory acts as secondary storage.

CHAPTER 3

DESIGN CONSIDERATIONS

A table is a two dimensional structure consisting of a set of cells. A cell is the smallest unit of the table and is addressable by specifying the row and the column number. It is separated from other cells by horizontal and vertical lines. The table interface and editor allow a user to create relations in a relational database and to modify its data. A suitable structure is needed to represent a relation. The tabular structure is an obvious choice as, the various attributes and tuples present in the relation can be represented by the columns and rows respectively. Moreover it provides a symmetric and systematic view of the data stored in the data base.

3.1 OBJECTIVES

(a) The table interface and editor must be able to perform the following tasks. It must allow the user

i) To define a table (i.e., specify the format of the table and get it displayed on the screen) in order to enter the various tuples of a relation.

ii) To modify a tuple.

iii) To modify the display format of the table.

iv) To define different tables representing the same relation in order to view desired attributes of the selected tuples

present in a relation. These kinds of tables are termed as views (or splittings) of the relation.

v) to define tables for different relations and get displayed.

(b) As far as possible, the function of various keys present on the key board must remain same as normal.

(c) System should be easy to learn.

3.2 DESIGN CHOICES

To perform the above tasks by the table interface and editor system the various design choices are made. They are as follows :

1. Table-interface is a screen oriented system because in this system it is desired that the table should be displayed on the terminal screen. Demarkation lines between rows and columns of the table should be of such characters that do not normally occur in the cells. For example character can be either '*' (to represent rows and columns) or dashes ('-', for rows) and bars '|', for columns). We have chosen bars and dashes.

A table can be defined in a number of ways. One way is to define it visually, Here first we put the cursor at the desired point on the screen and start making rows and columns of the table by typing dashes and bars.

Another way is to specify the row width and column width for each cell, System calculates the position of rows and

columns, of the table, on the screen and displays the table structure by displaying the bars and dashes accordingly. As the first method involves large amount of typing, so it needs more labour and time than the other. We have chosen the second method.

Names of the attributes should be placed at the top of each column of the table. They may be specified before the display of the table or after the display by typing each name at the top of the column. The first method has the limitation that the user can change the name of the attribute only after going through the steps involved in defining of a table. While in the second method, one will have to put the cursor in the cell (containing the attribute name) and modify it. Hence we have chosen the second method.

The key attributes of the system are those attributes whose values determine a tuple in the database uniquely. We decided to represent a key attribute by entering '*' character at the starting of that particular attribute name. The sequence in which the attributes are made as key attribute determine the order of preference given to these attributes during sorting. Any attribute can be deleted from or included in the set of key attributes by deleting or inserting '*' at the starting of that attribute name.

2. To fill a cell, one has to put the cursor in the cell. The movement of the cursor in the table should be free i.e. one should be allowed to move the cursor to any cell of the table. We have chosen arrow keys for this kind of movement as

Moveup : for moving the cursor up by one cell
Movedown: for moving the cursor down by one cell
Moveright: for moving the cursor right by one cell
Moveleft: for moving the cursor left by one cell.

If the cursor is present in the boundary (first or last) row or column (or both) of the table, arrow keys should scroll the table in the corresponding direction so as to display the next hidden cell roughly in the middle of current

i) row in case of moveleft or moveright, or
ii) column in case of move up or movedown arrow keys.

The hidden cell is thus seen along with some of the surrounding data.

To modify the contents of a cell characters need to be inserted or deleted. Insertion is performed by putting the cursor at the desired position and typing the desired character. Deletion of a character is done by placing the cursor at the position of the character and pressing the delete key. So the ability to move the cursor within a cell is essential. The cursor can be moved to the previous character by backspace key and to the next character by ^A.

To enter informations in cells within a row, the cursor has to be moved to each of the cells in the row. This involves, a lot of hand movement as the arrow keys are located far away from the other keys. Moreover, one would have to decide the proper arrow keys and their sequence for a desired movement. We have chosen the return key to avoid the above difficulties. Return key places the cursor in the next available cell of the same row. If it is the last cell of the row, then the cursor is put at in the first cell of the next row.

Sometimes need may arise to enter or modify an attribute value for a number of tuples. So there is no need to move the cursor through all the cells of the row. In such a case return key should place the cursor in the cell given by the next row in the same column.

If one wants to modify or enter some selected attributes of a number of tuples in a desired sequence, then he must first specify the sequence of column numbers. The return key now puts the cursor in the cell whose column number is next in the sequence in the same row. If it is the last cell of sequence, then the cursor is put in the first cell of the sequence in the next row.

Such movements are known as following a predefined path. Return key steps as through the cells in the paths.

A command is needed to define the paths.

3. There should be a command for altering the format of the table i.e., to change the numbers of rows, number of columns, row width and columns width. This command is helpful in defining the views (splittings) of a relation.

4. Additional commands that are needed are as follows :

(i) One may want to view a tuple in the relation which is not present in the table. As the table can accommodate very few tuples of data base at a time. There should be a command in order to display the desired tuples in the relation.

(ii) In case there are more than one table on the screen, the user may wish to move the cursor to any of his desired tables in order to do an operation on the data stored in the corresponding relation. There should be a command for moving the cursor to an appropriate table.

(iii) In the present implementation, the user may have sixteen tables (4 relations and each having 4 splittings) at a time. But, if the widths of rows and columns are large, then he may not be able to see all of them at a time. There should be a command so that user may have the display of his specified set of tables on the screen.

(iv) There should be a command to delete a tuple from the relation.

(v) There should be a command to remove a table (i.e. delete the table format) or a relation (i.e. delete the data along with all the tables corresponding to that relation).

There might be some additional commands to facilitate some of the operations. Such commands might be

(i) To redisplay the present screen contents

(ii) To save the data in the current session in order to preserve it. (As we are simulating the secondary storage in the main memory.)

(iii) To load a session from secondary storage

(iv) To print a table

(v) To ask for Help and quit the session.

5. Brief Summary should be displayed all the time. A summary line is needed to display the given command and the corresponding dialogue (including the error message in case of an error). Another line is needed to display the overflow string from a cell.

It looks natural to display the overflow string from a cell below the table containing that cell. Moreover, all the summary lines should be displayed together. Therefore, summary lines are displayed on the bottom of the screen.

3.3 USER MANUAL

Various commands available in the system are listed below.

On-line Commands

- Help Commands
  - (i) ? - for various help commands
  - (ii) ^V - help for various slash commands
  - (iii) ^U - help for update commands
  - (iv) ^P - help for path setting
- Inserting text
- Moving about on the table
- Going into slash mode

Various slash commands are as follows. These follow the slash ('/') character.

(i) C - creation of a relation

(ii) V - creation of a view

(iii) MC - moving the cursor to a particular cell of the present table

MT - moving the cursor to a desired table

(iv) T - displaying the desired set of tables

(v) F - find a tuple

(vi) D - delete a tuple

(vii) k - kill or delete a relation or a splitting

(viii) W - path setting for a desired type of movement

(ix) a S - saving a relation in a default filename

b ^S - saving a relation in a desired filename

(x) a L - loading a relation from a default filename

b ^L - loading a relation from a desired filename

(xi) P - print

(xii) R - redisplay the present contents of the screen

(xiii) Q - quit

(xiv) O - open

(xv) X - close

3.3.1 When the user runs the system by typing EX V.NEW/PAS, A,INP) the 'TITLE BOX' (indicating the name of the system) appears on the screen and the system waits for an input from the user.

Help Commands : Various Help commands are as follows :

(i) ? - Describes various Help commands

(ii) ^U - Help for update commands

(iii) ^P - Help for path setting

(iv) ^V - Help for slash commands

After displaying the information for any of the help desired by the user, the system waits for further input.

Update Commands : Now the user can create a relation either by /C or by /L, to be described later. After creating a

relation, cursor is put in the first cell of the table. It can be moved to any cell by arrow keys. An arrow key moves the cursor to the next cell in the corresponding direction.

When user types a character (other than ^P,^U, ^V,?, →, ←, ↑, ↓), it gets displayed and inserted at the position of the cursor and entered into the attribute of the tuple corresponding to that cell. Position of cursor determines the position of the character to be entered. Contents of a cell can be modified using the following commands.

(i) Control A (^A) : This moves the cursor to the next available character in the cell.

(ii) Backspace (^H or backspace key) : This moves the cursor to the character immediately preceeding the current position of the cursor in the cell.

(iii) Delete ('Delete' key) : It deletes the character preceeding to the current position of the cursor.

(iv) Control X (^X) : This moves the cursor to the last character of the cell from anywhere in the cell. This type of movement may be needed when appending to the contents of a cell is desired.

(v) Control R (^R) : It blanks the entire cell (i.e., writes 'blanks' everywhere in the cell) and Nil data is put into the attribute of the tuple corresponding to that cell.

(vi) Control F (F) : This searches for the specified character in the cell and moves the cursor to that character, if found.

If the cursor is present in the boundary (first or last) row or column (or both) of the table, typing arrow key command will scroll the table in the corresponding direction so as to display the next hidden cell roughly in the middle of the current

(i) Row in case of Moveleft or Moveright, or
(ii) Column in case of Moveup or Movedown arrow keys.

The scrolling provides to view different parts of a relation by the corresponding table. In the logical sense, a table is a window through which one can view a desired part of the relation. The position of a window on the screen is the viewport.

3.3.2 Slash Commands

These commands are given by typing '/'. The system responds with the message ('S', A,M,C,F,D,W,L,P,K,R,Q'). Now the user can type any of these characters according to his need.

1. Create Command : This command allows the user to define a table corresponding to the relation he wants to create. Command is given by'/C! System responds with the following

sequence of queries (when we say sequence of queries, it means next query in the sequence is displayed after the user has answered the previous one.).

(i) 'Enter Relation No.' - Here any integer no. (from 1 to 4) can be entered.

(ii) 'Enter no. of attributes'. Here user enters the no. of attributes (no. of attributes should be $\leq$ 10) in his relation.

(iii) 'Enter name of relation'.

(iv) 'Enter window-rows and window cols', i.e., size of table in terms of numbers of rows and columns.

(v) 'Enter row width'. In the present implementation all the rows of a table have several width. So only one row width needs to be specified.

(vi) 'Enter column width'. User specifies the widths for each column.

(vii) 'More relations? <Y/N>'. If user wants to create more relations he types 'Y' otherwise 'N'. In case of 'Y', the above mentioned sequence is repeated. In case of 'N' first screen gets cleared off and these table(s) is (are) displayed on the screen. Two lines are displayed below each table. One line indicates the relation number and splitting no. and other indicates the

relation name. After display the cursor is put in the first cell of the top row (this row is reserved for attribute names and its width is always 1). Now the user can start entering the attribute names. Any attribute can be made as key attribute by typing '*', as the first character. At a time, there can be at the most 6 key attributes. Order of priority given to these key attributes, during sorting, is determined by the sequence in which they are made as key attribute. Set of key attributes can be altered by deleting or inserting the '*' at the starting of the corresponding attribute names.

2. Split Command : This command allows the user to define the desired number of splittings (at the most 4) for a relation and also to alter the format of the existing tables. This command is given by /V. System responds with the following sequence of queries.

(i) 'Enter relation number'

(ii) 'Enter splitting number' - Any integer (from 1 to 4) can be entered.

(iii) 'Enter window-rows and window-cols'

(iv) 'Enter rows widths and columns width'

(v) 'More splittings? <Y/N>'. If more splittings are desired 'Y' is pressed otherwise 'N'. If 'Y' is pressed, the above sequence of queries is repeated. If 'N' is pressed then all the defined splittings of all the created relations, which can be accommodated on the screen are displayed.

3. Move : It allows to move the cursor to any cell of the current table or to the first cell of any desired table on the screen. Command is given by '/M', system responds with the message.

If 'T' is pressed, system asks 'Enter relation no.'. After entering the relation no., it asks 'Enter splitting no.'. User then enter the splitting no. Cursor is moved to the first cell of the table specified by the relation no. and splitting no.

If 'C' is pressed then system enquires 'Enter (row, column)'. The user enters the desired row no. and column no. on the current table and the cursor is moved to that cell.

4. Displaying the desired set of tables : Sometimes all the existing tables may not be accommodated on the screen (due to their size and number). This command allows to view a desired set of tables. Command is given by System responds with the following sequence of queries.

i) Enter relation no.

ii) Enter splitting no.

iii) 'More splitting? <Y/N>. If 'Y' is pressed, the whole sequence is repeated. If 'N' is pressed then system displays the tables specified in this command. If a table specified here is not existing then along with displaying the other tables it also displays the message that such and such table is not present.

5. Find: This command is given by'/F'. It is used to retrieve a specified tuple. System responds with.

'Give the key attributes in the order .,,..,..? one should enter the key attribute values in the same order as specified by the message.

In case the specified tuple is not present in the database, the system indicates it by, 'Tuple is not present', otherwise, the tuple is displayed on the screen. Cursor is put at the first cell of the row specified by the tuple.

6. Delete a Tuple : This command is given by 'D'. It allows to delete a specified tuple from the data base. System enquiries 'Give the key attributes in the order, ..., ...'. User enters the set of key attributes in the order indicated by the message and also the no. of tuples to be deleted. System deletes the specified no. of tuples starting from the specified tuple and gives the message 'No. of tuples deleted ...'.

7. Kill : This command is given by '/K'. It deletes the data and the format of the table corresponding to a relation from the secondary storage. System enquiries, 'Enter relation no.'. After entering the relation no. the message 'Relation is Destroyed' appears on the command line.

8. Way or Path : This Command is given by '/W'. It allows one to use RETURN key to follow a path. The system responds with the message 'Enter desired path-code'. The user may enter 2,3 or 4 according to a desired path movement.

(i) PATH CODE 2 : Path 2 allows to move in the next cell. It is useful to enter a large amount of data. After this Command is given, the system moves the cursor to the first cell of the table. Path 2 is the default path of the system.

(ii) PATH CODE 3 : Path 3 helps to view the data in a relational database. It allows to move the cursor in the same attribute present in the next tuple. If it is the last tuple, the system responds with 'No other tuple is present in the database' and path-code is automatically switched to 2.

(iii) PATH CODE 4 : It allows to move the cursor according to a selected attribute sequence. The system responds by the message 'Define Path'. The user should enter the attribute numbers in the desired sequence, i.e., if he wants to move along a path consisting of fourth, first and third attribute. He should enter the numbers like 4$1$3$$. Therefore, the cursor is positioned in the cell defined by first row and fourth column of the table. Typing the return key will place the cursor in the first cell of same row. Similarly, one can finally position the cursor in third cell of same row. On giving return, again, the cursor will be placed at fourth cell of next row. The above sequence repeats until the user give return from the cell defined by last tuple's third attribute. The path-code automatically becomes 2.

9 SAVE

There are two commands for saving the data of the current session along with the definition (or format) of a table.

(a) /S : This command is given by typing /S, system responds with the following sequence of queries.

(i) File : Here user enters the name of the file into which he wants to save the data. This file is known as data file. File name may have an extension (of maximum 3 characters). In case no extension is given by the user a default extension 'DBS' is assumed by the system. Definition (or format) of the table (to be specified in the following queries) is stored in a file whose name is same as that of data file but extension is 'FRM'. This file is known as format file. e.g. If data file is specified as ABC.D,then data will be stored in ABC.D and the definition of the table will be stored in ABC.FRM. If data file is specified as STUDNT, then data will be stored in the file STUDNT.DBS and definition of the table in the file STUDNT.FRM.

(ii) 'Enter relation no.'

(iii) 'Enter splitting no.'

Relation no. and splitting no. specify the table whose definition is to be stored.

After answering the above querries, the message 'saving started' appears on the screen. When saving is completed the message 'saving completed' is displayed.

(b) /^S : This command is given by typing /^S. This command is also used to save the data of the current session along with the definition of a table. But here the definition of the table is stored in a file (called format file) specified by the user. Sequence of queries is as follows :

(i) 'Enter data file name' : User enters the file name into which he wants to save the data. As in the previous case, here also if no extension is given,a default extension 'DBS' is assumed by the system.

(ii) 'Enter format filename' : Here user enters the file name into which he wants to save the format or definition of the table (to be specified in the following querries).

e.g. If data file is specified as ABC.D and format file as XY.Z, then data and format will be stored in these files respectively. If data file is specified as ABC and format file as XY, then data will be stored in the file ABC.DBS and while format will be stored in the file XY itself.

(iii)'Enter relation no.'

(iv) Enter splitting no.

After this the message 'saving started' appears on the screen and when saving is completed the message 'saving completed' appears on the screen.

10. Load : This command is used to bring the data back into the simulated secondary storage and the definition of the table in the main memory from the files saved as above. Like 'Save' here also we have two load commands.

(a) '/L' : This command is given by typing '/L'. This is used to load the data (the definition of the table from the files saved by '/S ' command. Sequence of queries is

(i) 'File' : User enters the data file name only. If the file name has the extension other than 'DBS', then he should specify the extension also.

When loading is completed the message 'loading over' appears on the screen.

(b) '/^L' : This command is given by typing '/^L'. This is used to load the data and the definition of the table from the files saved by '/^S' command sequence of queries is

(i) 'Enter data file name' : As above user enters the name of data file.

(ii) 'Enter format file name': Here he enters the name of the format file. If file name has extension other than 'FRM; then he should specify this extension also.

The message 'loading over' appears on the screen after

11. Print : One can take the hard copy of the data of a relation, in a tabular form. This command is given by typing P'. System responds with the following sequence of querries

(i) 'File' in which the data in the tabular form is to be stored

(ii) 'Enter table specs for each output page' and Enter relation no'.

Here user enters the relation no. for which the hard copy is desired.

(iv) 'Enter window-row and window-col'

(v) 'Enter row-width'

(vi) 'Enter columns width'

(vii) 'Do you want to have rows, columns numbered <Y/N>'.

It rows and columns on the hard copy are to be numbered, then 'Y' is pressed. Otherwise 'N'.

(viii) 'More pages? <Y/N>'. If the user wants to have the print-out with changed structure of the table on a separate page, then he presses 'Y' and again enters into the above sequence of querries. Otherwise he presses 'N'.

12. Redisplay : The command is given by '/R'. This allows to redisplay the present contents of the screen.

13. Quit : To quit from the execution one may press '/Q'.

14. OPEN : The user has to open a relation for doing any legal operation on the relation. This command is given by '/O'.

15. CLOSE : This command disables us to view or change a relation. It is given by '/X'.

3.3.3 Error Reporting

(i) Error Messages and their Meaning

If one, commits an error in giving a command or pressing a relevant key the system responds with an error message to indicate the error. It rings the bell also to draw the user attention. Following are the various error messages.

(1) 'Cannot move further up' in case the cursor is already present in the top row of the table and user presses '↑' key.

(2) 'Cannot further down' on pressing '↓' key when the cursor is in the bottom row of the table and the row is the last row of the W-sheet.

(3) 'Cannot move further left: when the cursor is present in the first column of the table and '←' key is pressed.

(4) 'Cannot move further right' on pressing '→' key when the cursor is present in the last column of the table and this column represents the last attribute of the relation.

(5) 'Character string can't be more than 100 characters long', in case when the no. of characters in a cell exceed 100.

(6) 'Tuples with the same key attribute is already present', when one tries to enter a tuple with key attribute value same as one already present in data base.

(7) 'No other attribute can be made as key' when there are already 4 key attributes in a relation and one tries to make one more attribute as key attribute.

(8) 'Attribute is already a key', when one tries to make a key attribute which is already a key.

(9) 'File is not present in the directory'. When, during loading (/L) one gives the file name which is not present in the directory.

(10) 'Relation is already present' when one wants to create a relation which is already present.

(ii) Errors Causing Monitor Intervention

There are some errors which cause the monitor intervention.

(1) If the user tries to load an arbitrary file in his directory (which has not been saved by the Table Interface), the monitor intervention occurs with the message 'scalar out of range'.

(2) The present implementation simulates a part of secondary storage in main memory. So if the main memory requirement crosses the maximum allotted quota, then the monitor intervention occurs with the message.

'Stack over runs heap, retry with more core'.

CHAPTER 4

IMPLEMENTATION

4.1 DATA STRUCTURE

1. Tables :

This data structure is used to store the definition of tables in main memory.

As defined in Chapter 3 a table is a set of cells. A cell is the smallest unit of the table that can be addressed by specifying row and column number and is separated from other cells by horizontal and vertical lines called delimeters

Minimum specifications required to define a table are :

i) number of rows and columns in the table

ii) starting row number and column number

iii) width of each row and column

iv) screen position of a distinct point (say left-top-most corner) of the table.

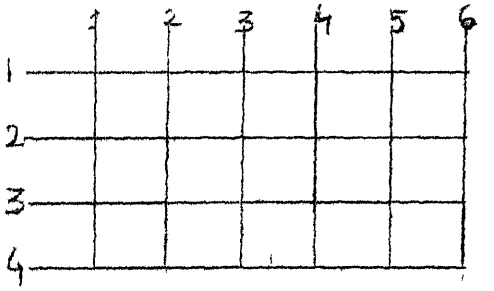


Fig (i)

A table is shown in Fig. (i). Here number of rows and columns are 4 and 6 respectively. Starting row number is 1 and starting column number is 1.

During movement from cell to cell or within a cell or during display of characters in a cell it is required to know the screen position of the boundaries of the cell. We can calculate these screen positions from the above specifications. But these screen positions are so frequently used that it is better to calculate them once and store into main memory than to calculate each time. As a column of the table always corresponds to a particular attribute of a relation, we can have different column widths depending upon the attributes of the relation. But the number of tuples in a relation may be large, so it is difficult to specify widths for all the rows (corresponding to tuples). Thus, we have same width for each row.

The data structure TABLES, storing the definition of tables is as follows :

```
Tables : array [1 ... Maxrels, 1 ... Maxtables] of record
         LM,RM:W-Cell;
         Columns:array[LPTCOL] of LPTCOLS;
         Rows:array[LPTCOLS] of LPTCOLS;
         Column-Width=array[1 ...W-Maxcols] of TTYCOLS;
         Width: TTYROWS;
         Window-Row:1 ... W-Maxrows;
         Window-col:1 ...W-Maxcols;
```

```
W-Cell = record
            row:0 ... W-Maxrows;
            col:0 ... W-Maxcols;
         end;
D-Cell = record
            row,col=LPTCOLS
         end;
```

TTYCOLS = 0 ... 100;TTYROWS=0 ... 24; LPTCOLS=0 ... 132.

W-Maxrows is the maximum number of tuples and W-Maxcols is the maximum number of attributes that one can have in a relation.

Here Window-Row and Window-Col is used to store the number of rows and columns in the table respectively. Array column-width stores the width of each column and width stores the width of a row (which is same for all the rows). LM is of type W-cell and stores the starting row number and column number of the table. Ld-Postn is of type D-cell and stores the screen position of the left-topmost corner of the table.

Last row number and column number is calculated from the LM, Window and Window-Col and is stored in RM. Screen position of each row and each column is calculated from Ld-Postn, Width and Column-Width and are stored in the arrays Rows and Columns. Rows [1] and Columns [2] together specify the screen position of the starting point of the cell addressed by first row and second column. Rd-Postn stores the screen position of the right-

bottom most corner of the table calculated from Rows, Columns, Column width and width.

2. VALREC :

As the system uses the relational data model, it is necessary to enter and retrieve the data at the tuple level. A tuple consists of various attribute values. An attribute value is a string of characters and so can be stored in an array STRING of characters.

```
STRING = packed array [TTYCOLS] of char,
         TTYCOLS = 0 ... 100 :
```

To store a tuple in main memory temporarily VALREC data structure is used. It is an array of type STRING. Here the attributes of the tuple are the elements of the array in order.

```
VALREC = array [1 ... W-Maxcols] of STRING ;
```

The variables of type VALREC (i.e. VALRECORD, VALRECORD 2) are used to transfer the data from the main memory to secondary storage and vice-versa.

3. KEY-REL and KEY-COL :

The present implementation allows us to have multiple key attributes. In this the searching and sorting will be done according to the sequence of the key attributes. Sequence of key attributes determines the order of priority given to each of these during searching and sorting.

e.g. Suppose there are 3 key attributes A,B and C. So, if their sequence is B,C,A then the comparison between two tuples will be made first on the values of B then on the values of C, and lastly on the values of A.

The above shows that there is a need to store the sequence of the key attributes. More memory is required to store the key attribute names than that required to store the number of each key attribute (each attribute of the relation is associated with a unique number). Memory requirement is reduced further if the numbers are stored in the form of characters. An array key-Rel of type character can be used to store the number of each attribute present in the sequence. Position of this number in the array determines the order of priority given to the corresponding the key attribute.

```
Key-Rel = packed array [1 ... 7] of char,
Key-col = array [1 ... Maxrels] of Key-Rel
```

Maxrels stores the maximum number of relations that one can have in secondary storage.

Considering the above example if A,B and C be the key attributes of relation No. 1 and the numbers associated with these are, say, 3,6 and 8 respectively, then a sequence B,C,A will be stored as

Key-Col [1][1] = CHR (6)

Key-Col [1][2] = CHR (8)

Key-Col [1][3] = CHR (3)

As the first element of the array is CHR (6); so, while comparing 2 tuples, first preference will be given to the corresponding key attribute, i.e., 'B' and and so on.

4. PATT-ATT :

In case of path 4 (ref. Chapter 3) user has to specify the sequence in which he wants to move the cursor through the desired attributes. So it is required to store in the main memory, these desired attributes and the corresponding sequence. An array Path-Att of integers can be used to store the number of each of the desired attribute. Positions of these numbers in the array will determine the sequence.

Path-Att : array [1 ... W-Maxcols] of integer;

For example, if the user wants to move the cursor in the sequence 2nd, 4th and 3rd attribute, then these numbers (i.e., 2,4 and 3) will be stored in Path-Att as follows :

Path-Att [1] = 2, Path-Att [2] = 4, Path-Att [3] = 3 .

5. ROW-KEYS and KEYS :

A relation has a number of tuples, so finding a particular tuple may require a large number of accesses to secondary storage. One way to reduce the number of accesses is to store the

relation in an ISAM file and build an Index. As we are simulating secondary storage in main memory, we can store only a few tuples. So an Index can be simulated by an array Row-Keys of type keys which stores the key attributes of each tuple of the relation (As shown below keys store r all the key sttributes of a tuple).

```
Row-keys = array[1 ..W-Maxrows] of Keys;
    Keys = array[1...Max-Keyatt ] of STRING.
```

Max-Keyatt is the maximum number of Key attributes that one can have in a relation.

Further key attributes of the tuples, can be stored in the same order as the tuples in the simulated secondary storage. Therefore, the position of key attributes of a tuple in Row-Keys determines the position of that tuple in the secondary storage.

6. Rdset-Type, Tdset-Type, Tabset

In the present system a user can have at the most 4 relations (Maxrels) and for each relation maximum number of splittings (views) is 4. So at a time he can have at the most 16 tables. A variable is required to store the number of relations that user has created and another variable is required to store the numbers of all the defined splittings of each of the created relations.

Numbers of created relations and numbers of defined splittings can be stored in the variables of type Rdset-type and Tdset-type respectively.

```
Rdset-type = set of 1 ... Maxrels
Tdset-type = array [1 ... Maxtables] of Tabset
Tabset     = set of 1 ... Maxtables.
Relset, Rdset: Rdset-type,
Tblset, Tdset: Tdset-type .
```

If one has created relation No.s 1 and 2 and for relation No.1 he is having two splittings 1 and 2, and for relation number 2 the splittings are 2 and 3, then

```
Relset = [1,2] and Tblset [1] = [1,2]
                   Tblset [2] = [2,3]
```

Sometimes screen can't accomodate all the tables defined by user. In such a case variables Rdset and Tdset are used to store the relation number and corresponding splitting number that the user wants to see on the screen.

Thus considering the above examples, if out of 4 tables user wants to see only splitting no. 1 of relation no. 1 and splitting no. 3 of relation no.2 then

```
Rdset = [1,2] and Tdset [1] = [1]
                  Tdset [2] = [3] .
```

7. W-Sheet

A relational data model can be thought of as a two dimensional structure in which one dimension (say horizontal) specifies a tuple (a row) while the other (say vertical) specifies an attribute. So an obvious choice for storing a relation is a two dimensional array. One dimension, called row, specifying a tuple and other called column, an attribute. Both dimensions together specify the value of an attribute. As the value of an attribute is a string of characters, it can be stored in a variable of type STRING. To have more than one relation one more dimension is needed. So the final choice for the data structure storing the relations is

```
W-sheet: array[1 ... Maxrels, 0...W-Maxrows,
                1 ... W-Maxcols] of ^STRING .
```

Instead of STRING, here we have used ^STRING to avoid unnecessary allocation of memory.

The array W-sheet is used to store the data of a relation and the name of all attributes. For a relation, data is stored in rows from rowno = 1 to rowno = W-maxrows. The name of attributes are stored in rowno = 0. These are stored in the same way as the data is stored in W-sheet.

The functional block diagram illustrates the various controls and corresponding flow of data in the system. The

FUNCTIONAL BLOCK DIAGRAM

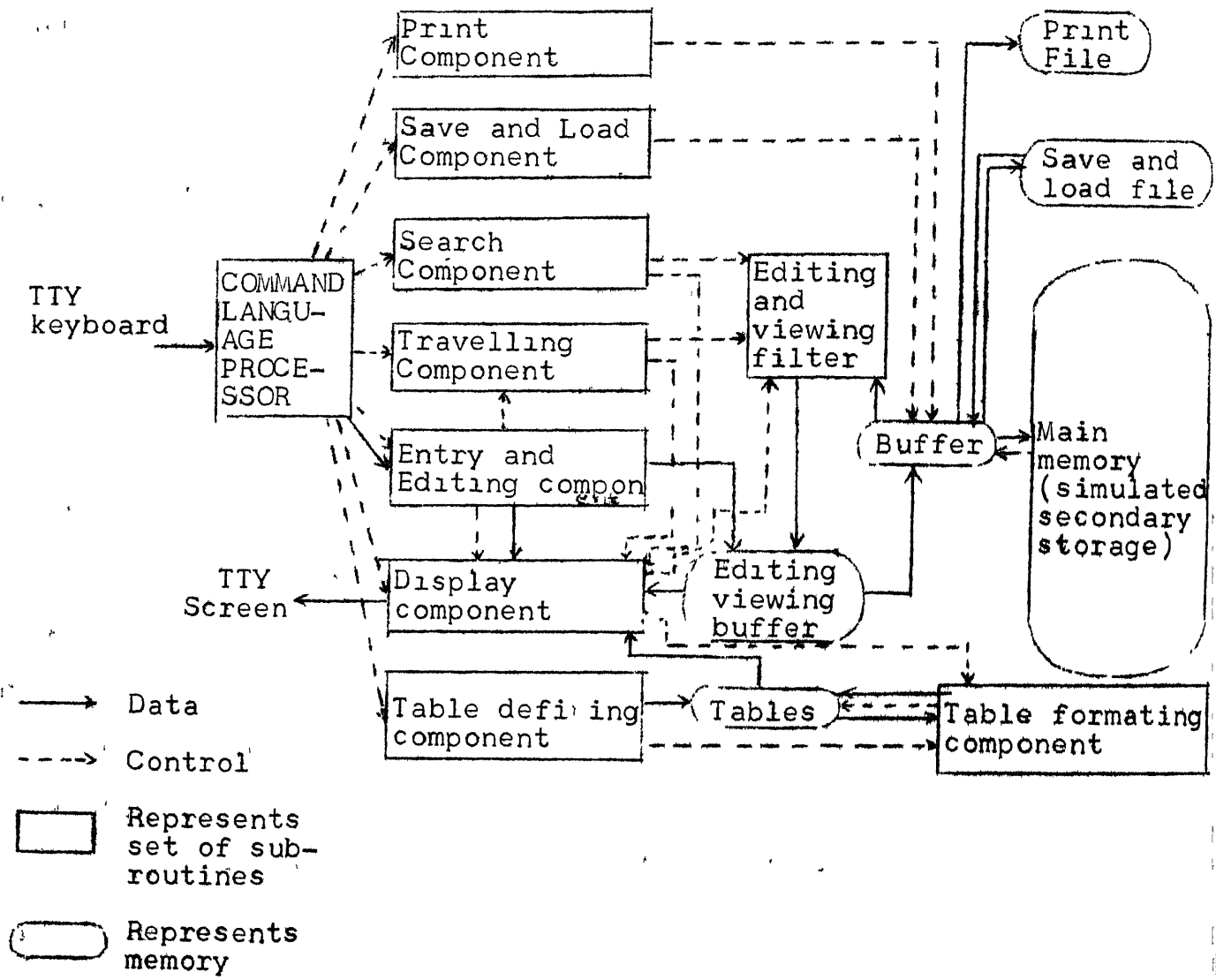


Fig. (ii) A

entered string of characters is recognized by the COMMAND LANGUAGE PROCESSOR and further action is taken depending on the type of input string. It can be a data or a command. The data is entered at the tuple level through the editing and viewing buffer invoked by entry and editing component. The data is simultaneously displayed on the screen by 'Writechar' (Ref. Fig. (ii)) procedure of Display component.

If the entered character or string is a command, the COMMAND LANGUAGE PROCESSOR' invokes the following appropriate component for its processing.

i) The Table defining component is used to define the format for a desired table. It is invoked by /C or /V commands. /C is used to define a table corresponding to a relation and /V creates a splitting (view) for a relation. This component invokes Table formatting component to fill the various fields of Tables data structure record (Ref. page 4.1). The Table is finally displayed by the Display component.

The Table formatting component can be invoked by the Display component so as to modify the format of table during search or travelling.

ii) The search component enables to find a particular tuple from the database and display it on the screen. It generates editing and viewing buffer after processing through Editing and Viewing Filter. Then, the contents of editing and viewing buffer are displayed by display component.

DISPLAY COMPONENT

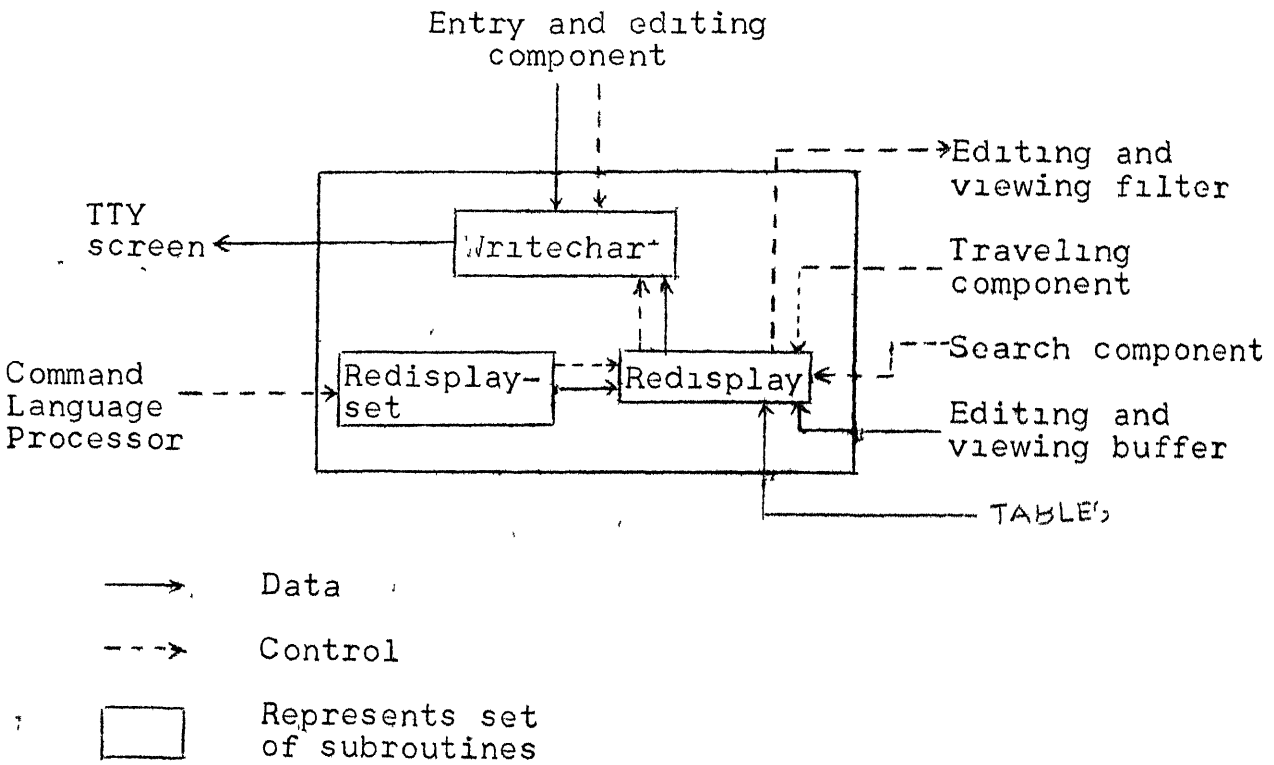


Fig. (ii) B

iii) The Travel component enables to move the cursor on a table. In case of scrolling, it displays the hidden part of the Viewing buffer by implicitely invoking the Display component. While following a movement in PATH 3 or PATH 4, it generates editing and viewing buffer in the case of a new tuple to be displayed.

iv) As discussed above, the Display component can be implicitely invoked by Entry and Editing, Traveling or search component. One can explicitely invoke the Display component by typing /R Command. This command redisplays all existing or desired tables. As the editing and viewing buffer can store a single tuple at a time, so the data of the table (data corresponding to 2 or more tuples) is displayed by filling the editing and viewing buffer and displaying it until the editing and viewing buffer contain the last tuple of table. In this case the Display component invokes the Editing and Viewing Filter for filling the editing and viewing buffer with the tuples corresponding to the data of the table.

v) The print component enables to take hard-copy of a table. It invokes Buffer (ADRS). The desired table data is transferred from simulated secondary storage to print file via Buffer.

vi) The Save and Load component enables to save and load the desired format of a table or data of a relation. The save and

load component invoke Buffer. The desired format of a table or data is transferred from simulated secondary storage to save file or vice-versa (in load).

It is desired that there should be no separate command to go into edit-mode (editing commands are entered like data) and the editing of a form should be visual. So, editing buffer and viewing buffer should have the same data. In our system, the variables VALRECORD of type VALREC represents the editing and viewing buffer.

4.2 PROGRAM STRUCTURES

The above described components consist of various sub-routines. As the system is interactive, it needs to take input from the user. The procedures INP and READNO are used to input a character and an integer simultaneously. Moreover, the system is screen-oriented, so the cursor positioning on the screen is required. This facility is provided by PUTCURSOR. The description of these subroutines is given below.

1. INP (Var Ttyinp:integer);

Function : It reads the character (input from TTY keyboard) and stores the ASCII code in the variable ttyinp.

2. READNO (Var I:integer);

Function: It reads an integer and stores it in variable I.

Global variables accessed: Eoln, C-col, ttyinp.

Global variables modified: Eoln, C-col, ttyinp.

3. PUTCURSOR (Row, Col: integer);

Function : It positions the cursor at the point defined by integers row and col on the screen.

In the following part of this chapter, we will term it as 'cursor position'.

Global variables accessed : C-row.

Global variables modified : C-row, C-col.

The program structures for the various components of Fig. (ii) are listed below.

(A) The Table defining component enables us to define a table by /C or /V command. It fills tables record by INIT-TABLE procedure and invoke the Table formatting component. The Table formatting component consists of SETCOLS-ROWS and STARTING-POINT procedures. SETCOLS-ROWS calculates the screen position of various rows and columns and STARTING-Point calculates the screen position for the top left point of the tables record. The description of these procedures is as follows.

1. INIT-TABLE (Relno, Tableno: integer);

Function : It inputs the various parameters i.e. window-row, window-col, row-width and column-widths for tables [Relno, Tableno].

Global variables accessed: Eoln, Relat [Relno] Maxatt.

Global variables modified: Ttyinp, Tables [Relno, Tableno],Eoln.

2. SETCOLS-ROWS (Relno:integer; Tableno: integer);

Function : It calculates the screen position of each row and column of the tables [Relno, Tableno] and stores them into Rows and Cols array of tables record. Moreover, the procedure manages the right bottom point of the table so that it should be on the screen.

Global variables accessed : Tables [Relno, Tableno], Print, D-maxrows, D-maxcols.

Global variables modified : Tables[Relno,Tableno].

3. STARTING-POINT (Rset: Rdset-type; Tset: Tdset-type);

Function : It calculates the screen position for the top left point of each table, present in RSET, TSET. Also, it manages the possible set of tables which can be accommodated on the screen.

Global variables accessed : Rdset, Tdset, Tables.

Global variables modified : Rdset, Tdset, Tables.

(B) The Travelling component is used to move the cursor at the desired cell of a table. It consists of subroutines, MOVEUP, MOVEDOWN, MOVERIGHT and MOVELEFT. These are described as follows.

1. MOVEUP;

Function : It moves the cursor up by one cell. During scrolling, this procedure places row of the cell roughly in the middle of the new viewport.

Global variables accessed : Tables[Relindx, Tblindx], W-row.

Global variables modified : Tables[Relindx, Tblindx], W-row.

2. MOVEDOWN;

Function : It moves the cursor down by one cell. During scrolling, this procedure places row of the cell roughly in the middle of the new viewport.

Global variables accessed : Tables[Relindx, Tblindx], W-row.

Global variables modified : Tables[Relindx, Tblindx], W-row.

3. MOVERIGHT;

Function : It moves the cursor right by one cell. For the case when one moves out of the table, this procedure places the column of the cell roughly in the middle of the new viewport.

Global variables accessed : Tables[Relindx, Tblindx], W-col, Relat[Relindx] Maxatt.

Global variables modified : Tables [Relindx, Tblindx], W-col.

4. MOVELEFT;

Function : It moves the cursor left by one cell.

Global variables accessed : Tables[Relindx, Tblindx], W-col.

Global variables modified : Tables[Relindx, Tblindx], W-col.

C) The Search Component is used to find a desired tuple and display it on the screen. It consists of routines FIND-POSITION and FIND-TUPLE.

1. FIND- POSITION (Var J: integer);

Function : It inputs the Keyattributes of a desired tuple, finds the tuple in the W-SHEET.

(By comparing with key index [RelINDX]) and outputs J integer (equal to Rowno. in W-sheet) i.e., the address of the tuple.

IF tuple is absent then search-fail is assigned false

Global variables accessed: Keyindex [Relindx], key-col [Relindx], W-maxrows.

Global variables modified : Search-fail.

2. FIND-TUPLE;

Function : It finds and displays the desired tuple.

Global variables accessed : Search fail, Tables [Relindx, Tblindx], W-row, Keyindex[Relindx], Key-col [Relindx], W-maxrows.

Global variables modified : Tables [Relindx, Tblindx], W-row, W-col, C-row, C-col.

D) The display component is used to display a set of tables on the screen. The /R command invokes REDISPLAY-SET procedure of the Display component. This procedure calls REDISPLAY procedure. The REDISPLAY procedure displays a single table and the data of the table is displayed by WRITECHAR procedure.

1. WRITECHAR (CH: Char; Relindx, Tblindx: integer);

Function : It displays the CH character on the screen. The character is displayed in a cell or on the extension line if,

the cell is already full.

Global variables accessed : W-row, W-col, C-row, C-col, Tables [Relindx, Tbindx)

Global variables modified : C-row, C-col.

2. REDISPLAY (Relno: integer; Tableno: integer);

Function : It displays a table specified by Relno and Tableno. integers.

Global variables accessed : Tables [Relno, Tableno], W-sheet

Global variables modified : Valrecord, Rowl, C-row, C-col.

3. REDISPLAY-SET;

Function : It displays the present set of tables existing in the system . In case if a table is absent, then it gives the corresponding Error message.

Global variables accessed : Relset, Tableset, Rdset, Tdset.

Global variables modified : Rdset, Tdset, Dmysetl, Dmyset2, Blank-screen, C-row, C-col.

E) The Entry and editing component is used to enter or modify a string of characters. It invokes Travelling component implicitely to position the cursor at the desired cell, then the movement of cursor within a cell is done by move-ahead, backspace etc. component of the ENTER-INSERT-MODE procedure. Moreover, the entry, insertion or updating of a cell is also performed by this procedure. The deletion of a character is done by DELETECHAR procedure. The subroutines are given below.

1. DELETECHAR (Var String: line)

Function : It deletes a character pointed by the Pointer variable in a given string.

Global variables accessed : W-row, pointer, W-col, length, Key-col [Relindx], ROWDB [Relindx], W-sheet.

Global variables modified - Pointer, length, E-row, E-col, Key-col [Relindx], W-sheet, C-row, C-col.

2. ENTER-INSERT-MODE (Var Valre: Valrec; Var String: line);

Function : It allows to enter an input character and modify the cell (where the cursor is present) by extend, replace, search, delete, backspace, movement etc. edit commands.

Global variables accessed : Tables [Relindx, Tblindx], W-row, W-col, length, ttyinp, C-row, C-col, Nilattribute, path-code, Rowdb [Relindx], PP,Path-att [PP], Tupleover, W-sheet.

Global variables modified : PP, C-row, C-col, W-sheet.

F) The print component is used to get hard-copy of desired table. It invokes READFILENAME, PRINTING and PRNTFILE procedures. READFILENAME reads the desired name of file. PRNTFILE allows to print one table and PRINTING procedure is used to invoke PRNTFILE so as to print a desired number of tables. The description is given below.

1. READFILENAME (Var Filenm : Alfa);

Function : It reads the input characters and stores them into variable 'Filenm'.

Global variables modified : TTYinp.

2. PRNTFILE (Filename:Alfa; Relno, Tableno: integer);

Function : It fills a file with the contents of a table defined by Relno and Tableno.

Global variables accessed : Tables [Relno, Tableno], W-sheet.

Global variables modified : Numbering, delimiters, print.

3. PRINTING;

Function : It makes a file (for printing) by PRNTFILE subroutine.

Global variables accessed - Rdset, Tdset, Tables.

Global variables modified - Ttyinp, PRINT, Tables, Rdset, Tdset.

G) The save and load component allows to save and load the different formats of different tables in the different files. The saving and loading of data also provided by this component. The component consists of the subroutines SAVE-FORMAT, SAVE-DATA, LOAD-FORMAT, LOAD-DATA. The description is given below.

1. SAVE-FORMAT (A,I:integer);

Function : It stores the format of a table (specified by A and I integers) in a desired filename.

Global variables accessed : W-sheet, Tables [A,I], Relat[A]. maxatt.

Global variables modified : Temp.

2. SAVE-DATA (A: integer);

Function : It stores the data of relation 'A' in the Temp.file.

Global variables accessed : W-sheet, Relat [A]. maxatt.

Global variables modified : Temp.

3. LOAD-FORMAT

Function : It loads the format of a table from the Tempfile.

Global variables accessed : Temp, open-set.

Global variables modified : Relat [A],maxatt, W-sheet,Tables[A,I]

4. LOAD-DATA;

Function: It loads the data of a desired file in the W-sheet.

Global variables accessed : Temp.

Global variables modified : W-sheet, Row1, Row2.

H) As we are simulating the secondary storage in the main memory, so some procedures are simulated to provide the data flow from the simulated secondary storage to Buffer or vice-versa, as shown in Fig. (ii). GET-record and PUT-RECORD are used to transfer the data from W-SHEET (simulated secondary storage) to Buffer (ADRS) or vice-versa respectively. INSERTRECORD is used to insert a tuple (in the ascending order) in W-sheet. OPEN allows us to open a relation for any legal operation to be done on it. CLOSE procedure is used to close a relation.

In the actual implementation, all these procedures will be modified and the file access functions will be called by the corresponding present routines.

1. GET-RECORD (Chno:integer; Var Adrs:Valrec);

Function: It transfers a tuple located at Row1 of W-sheet (simulated sec storage) to Adrs. The tuple exist in relation numbered by Chno.

Global variables accessed : W-sheet, Row1, Relat.[Relindx]Maxatt

2. PUT-RECORD (Chno: integer; Adrs: Valrec);

Function : It transfers a tuple in Adrs buffer to Row2 location of W-sheet. The tuple exists in Chno relation.

Global variables accessed : Row2, Relat[Relindx].Maxatt.

Global variables modified : W-sheet.

3. INSERTRECORD (Chno :integer; Adrs:Valrec);

Function : It inserts the record of Adrs in W-sheet. The insertion is performed in ascending order by comparing the key-attributes in keyindex [Chno].

Global variables accessed : RowDB[Chno], Key-Col [Chno], Keyindex[Chno]

Global variables modified : Rowdb [chno], keyindex[chno], Row1, Row2.

4. OPEN;

Function : It opens a relation by adding [Relno] to open-set and set the values of Relat[Relno].maxatt and key-col[Relno].

Global variables accessed : Open-set, W-sheet, Relset.

Global variables modified : open-set, Relat[Relno].maxatt, key-col [Relno].

5. CLOSE;

Function : It closes the relation by substracting [Relno] from open-set and stores Relat [Relno].maxatt, key-col [Relno] in W-sheet and reset the values of these variables.

Global variables accessed : Relset, open-set, key-col[Relno], Relat[Relno],maxatt.

Global variables modified : W-sheet,open-set, key-col[Relno], Relat[Relno].maxatt.

CHAPTER 5

CONCLUSIONS

5.1 SUMMARY

Table interface and editor system based on relational data model has been implemented. Here, the user is allowed to enter, display and edit data through a table. The table is displayed on the ordinary terminal screen i.e., graphic terminal screen is not required. Data is entered by first positioning the cursor in the desired cell of the table and then typing the characters. Thus, the system is screen-oriented. The system allows a user

1. to define a table in order to enter various tuples of a relation.
2. to display and modify a desired tuple. The tuple is displayed roughly in the middle of the table along with the adjacent data.
3. to browse through a relation by scrolling the table. Both vertical and horizontal scrolling have been implemented.
4. to define upto four views (or splittings) of a relation.
5. to define upto sixteen tables representing the four views of four relations.
6. to display the desired number of tables on the screen.

7. to alter the set of keyattributes and modify their sequence of preference.

We have implemented paths, a facility by which the Return key steps us through the cells in the predefined path. Brief summary lines are displayed on the screen. These display the given command and the corresponding dialogue (including the error messages in case of an error). Thus, the system is interactive. It provides help at all times. The facility for

i) printing,

ii) saving into a file,

iii) loading from a file and

iv) redisplaying the present set of tables are also incorporated in the system.

5.2 LIMITATIONS

The following are the limitations in our system.

1. As the present system is screen-oriented, one has to position the cursor on the screen for entering or editing an attribute of a tuple. The direct-addressing of the cursor is a hardware feature of a terminal and the command for addressing the cursor may be different for different terminals we used the command for 'DECSCOPE-TERMINAL VT52A' in our system, so the system will work only on this terminal.

In case, one wants to use the system on some other terminal, he should replace the existing command by the actual command in PUTCURSOR procedure of the system.

2. While filling a row of a table, one can't enter first charater a '?' in first cell because '?' is used as the help charater.

One may enter '?' by first typing '^X', '^A' etc. and thereafter positioning the cursor at the first character place of the cell.

3. The specification of format of a table (corresponding to a relation) involves specifying desired total number of attributes for the relation. After specifying a table, one can't alter (i.e., increase or decrease) the total number of attributes.

4. A table represents a single relation in the present implementation, i.e., a table doesn't represent the multiple relations.

5.3 FUTURE-WORKS

The following are the extensions to our system suggested for the future :

1. The present system involves simulation of secondary storage. To make it an actual system, one should use actual secondary storage. The following routines for accessing the secondary storage are needed to be implemented.

(A) GETRECORD (Chno, Adrs, Keyval)

(B) GETEXTRECORD (Chno, Adrs)

(C) INSERTRECORD (Chno, Adrs)

(D) DELETERECORD (Chno, Keyval)

(E) UPDATERECORD (Chno, Adrs)

(F) MODIFYRECORD (Chno, Adrs)

(G) OPENR (Chno, Fname)

(H) CLOSER (Chno)

2. The present implementation can be extended to make a supercalc system. Here, table will become the supercalc worksheet. One should be allowed to enter an expression or a data in the cells of supercalc worksheet. The expression of a cell should be evaluated for the calculations in the system. So, an EVALUATOR program is needed to be designed.

In supercalc system, one may want to view the expression or the data of a cell. A command is needed to switch the mode of display, the modes may be expression or numeric.

3. Presently, the data is stored in the form of characters in the secondary storage. This may result in more memory requirement for storing the integers of large values. The extension to the present system is to store the integer data in integer form and real data in real form.

4. Presently, printing involves producing the copies of a table, one after the other. If the table is small, one may desire to have some copies at the empty side-portion of the page also. So, this feature can be incorporated in the system.

Further extensions are possible in the system depending on the needs of a user and the imagination of a programmer.

REFERENCES

1. Norman Meyrowitz Andries Van Dam, Interactive Editing Systems: Part I and II, acm Computing Surveys, Volume 14, Number 3, September 1982.
2. Ullman Jeffrey D, Principles of Database Systems, Galgotia Publications, 1984.
3. Donald H. Beil, Supercalc! The Book, Reston Publishing Company, Inc., 1983.
4. Shrivastava Vinay Mohan, Form Interface and Editor for relational database.